FreeRec: an Anonymous and Distributed Personalization Architecture

Antoine Boutet, Davide Frey, Arnaud Jégou, Anne-Marie Kermarrec,
Heverson Borba Ribeiro

FreeRec: an Anonymous and Distributed Personalization Architecture

Antoine Boutet[1], Davide Frey[1], Arnaud Jégou[1], Anne-Marie Kermarrec[1,2], and Heverson B. Ribeiro[1]

[1] INRIA Rennes, France
antoine.boutet@inria.fr, davide.frey@inria.fr, arnaud.jegou@inria.fr, anne-marie.kermarrec@inria.fr, heverson.ribeiro@inria.fr
[2] EPFL, Switzerland

Abstract. We present and evaluate FreeRec, an anonymous decentralized peer-to-peer architecture, designed to bring personalization while protecting the privacy of its users. FreeRec's decentralized approach makes it independent of any entity wishing to collect personal data about users. At the same time, its onion-routing-like gossip-based overlay protocols effectively hide the association between users and their interest profiles without affecting the quality of personalization. The core of FreeRec consists of three layers of overlay protocols: the bottom layer, rps, consists of a standard random peer sampling protocol ensuring connectivity; the middle layer, PRPS, introduces anonymity by hiding users behind anonymous proxy chains, providing mutual anonymity; finally, the top clustering layer identifies for each anonymous user, a set of anonymous nearest neighbors. We demonstrate the effectiveness of FreeRec by building a decentralized and anonymous content dissemination system. Our evaluation by simulation and through extensive PlanetLab experiments show that FreeRec effectively decouples users from their profiles without hampering the quality of personalized content delivery.

1 Introduction

The Web 2.0 has transformed the way users interact with the Internet. Users are no longer pure consumers, but they now generate a large portion of the available content. As a result, personalized services have become a requirement for most online applications. While personalization and social applications greatly enhance user experience, they amplify the Internet's inherent privacy risks and concerns. For instance, personalization in a social application can lead to the revelation of potentially embarrassing information to friends, family, and colleagues. In addition, users publishing controversial or prohibited information on social platforms can easily be identified and located through their IP addresses.

The reason for the privacy risks associated with personalized services lies in their inevitable dependence on personal data. As another example, consider one of the most common forms of personalized services: recommendation. A

common technology providing this service is user-based Collaborative Filtering (CF) [24]. This paradigm leverages interest similarities to identify correlations between the preferences of different users. While users are not generally aware of who else shares their own interests, their centralized implementation requires service providers to store accurate information about the interests of users. This clashes with the need to protect personal data.

Anonymity services provide an attractive way to overcome the privacy issues associated with personalized services. They hide the real identity (*i.e.* IP address) of a user through pseudonym (*e.g.* IP address of another node). Several such solutions are available on the Internet [1] and offer users the possibility to navigate anonymously behind a proxy. However, the use of a single proxy is vulnerable to adversaries that can observe traffic going in and out of the proxy. Distributed solutions, such as Tor [11] provide better guarantees. Nonetheless, they do not eradicate the concentration of personal data within the servers of a single provider. Decentralized personalization based on the P2P paradigm [5, 8] addresses the issue of concentrated data while providing naturally scalability. Yet, they remain vulnerable to the presence of malicious users.

Clearly anonymity alone does not protect users privacy, nor does decentralization alone. In this work, we seek to address these issues by combining the benefits of decentralized personalization and anonymity. The result is FreeRec, an anonymous and distributed personalization architecture. Our solution implements a distributed user-based (CF) scheme through an anonymous and interest-based topology and uses the resulting overlay to recommend items to users. Unlike existing decentralized personalization platforms, FreeRec protects the interest profiles stored at every node by means of anonymous exchanges with other peers. This makes FreeRec a generic personalization architecture that can be leveraged to build a number of distributed applications that may benefit from recommendation services.

FreeRec builds anonymous chains of nodes by relying on three layers of gossip protocols providing mutual anonymity. A standard random-peer-sampling protocol provides nodes with the members of their anonymous chains. A second private peer-sampling protocol uses these chains to provide each node with an anonymous sample of the network. A top clustering layer implements a decentralized collaborative-filtering overlay by creating decentralized clusters of anonymous profiles. This layered architecture makes FreeRec self organizing and capable to adapt to the arrival and departure of nodes and to changes in the interests of users. We evaluate FreeRec using both simulation and a real deployment on PlanetLab. Our results on a news-personalization use case show that users are able to effectively receive and publish content even in presence of path failure with reasonable overhead.

2 System model

We consider a decentralized user-based collaborative filtering (CF) system [5, 8]. Such systems build interest-based overlay networks by clustering nodes according

to the similarity among their interest profiles[3]. This task relies on two protocols: a random peer sampling (RPS) and a clustering protocol (CLUSTERING). The RPS [26] protocol ensures connectivity by providing each node with a continuously changing random sample of the graph. This comes in the form of a view data structure: a list of references to other nodes. Each entry in a view consists of (*i*) a node's IP address and port, (*ii*) a profile describing the node's interests, and (*iii*) a timestamp indicating when the associated profile was generated. While the RPS allows nodes to continuously discover new nodes, the CLUSTERING protocol identifies, at each node, the k-nearest neighbors in term of interests, and ensures connectivity between the node and this neighborhood.

Periodically, each protocol selects the node in its view with the oldest timestamp and sends it a message containing its profile with half of its view for the RPS and its entire view in case of the clustering protocol (standard parameters [14, 27]). In the RPS, the receiving node renews its view by keeping a random sample of the union of its own view and the received one. In the clustering protocol, it computes the union of its own and the received view, and selects the nodes whose profiles are closest to its own according to a similarity metric. Several similarity metrics have been proposed [25], we use the Jaccard index in this paper.

3 FreeRec

Our anonymous personalization architecture extends the model described in Section 2 to achieve anonymity by executing gossip exchanges through onion-like encryption chains: the *proxy chains*. The *proxy chain* of a node n is a sequence starting with n and containing a random number of other nodes – from $\mathsf{ch}_{\mathsf{min}}$ to $\mathsf{ch}_{\mathsf{max}}$ – as depicted in Figure 1. We refer to node n, the first node in the chain, as its *initiator*. The last, p, is the chain's *proxy*, (or n's proxy), while the remaining ones are *intermediate* nodes. Messages can travel along the chain in two directions: *forward*, from the initiator to the proxy; or *backward*, the other way around. The proxy acts as a placeholder for n, hiding n's identity in all the gossip exchanges that include n's interest profile.

Proxy chains effectively hide the very fact two nodes are communicating. Two nodes n and m can learn their respective profiles without knowing their respective identities. Moreover, their profiles are hidden from all other nodes in the chains. A node n that wishes to send a message to another node builds a sequence of encryption layers around it, including the corresponding routing information. Each of the nodes along its proxy chain removes one of these layers and sends the inner encrypted layer to the next hop indicated in the message. The process continues until the destination node's proxy. At this point, the message goes through the destination chain in the backward direction using routing information and encryption keys maintained by each node in the chain. Each of these, starting from the proxy, adds one encryption layer and routes the message until it arrives at the destination node, which removes all the layers.

[3] We use the term node to refer both to a user and to her machine.

3.1 Chain-Based Routing

We now present the data structures that allow nodes to build, maintain, and use proxy chains.

Chain and Message Keys. The onion-like encryption process outlined above relies on three types of keys: two sets of public/private key pairs, and one set of secret keys. First, each node, n, maintains a key pair, (K_n, k_n)[4], called *message key pair*. Nodes use it to send and receive encrypted messages through proxy chains to and from any other node while preventing the proxies and the other chain nodes from accessing the content of this communication.

Each node also maintains a second key pair: the *chain key pair*, (C_n, c_n). While the message key pair hides the content of a message from the nodes in the chain, the chain key pair makes it possible to construct the onion-like encryption layers when traversing the chain in the forward direction.

Finally, each node, n, generates and dispatches a secret key, s_i^n, to each node, i, in its own proxy chain. Nodes use this key to add onion layers to messages that travel along the chain in the backward direction, *i.e.* towards n. The use of onion-like encryption in the forward and backward directions causes messages to change at each hop, thus preventing external observers from recognizing the messages in a proxy chain. We summarize the roles of the three types of keys in Table 1, and provide details about their distribution in Section 3.3.

Data Structures and Routing IDs. To route messages along proxy chains we use a combination of source and hop-by-hop routing. Each node maintains information about the members of its own proxy chain in a CHAINTABLE. This data structure is essentially a list: each entry consists of the identifier of a node, and of its associated public chain key. The information in the table allows the initiator of a chain to encrypt messages in onion layers.

The destination proxy, however, cannot use source routing to reach the destination node: a node may in fact act as a proxy or an intermediate node in multiple proxy chains. To route backwards along the chain, we therefore use a set of ROUTINGIDS as depicted in Figure 2. For routing purposes, all the nodes in a chain could use the same ROUTINGID to identify their next hops. However, this would easily allow colluding nodes to verify if they are part of the same chain. We therefore associate a unique (with high probability) ROUTINGID with each link in a chain. The proxy ROUTINGID (e.g. p_a and p_b in Figures 1 and 2) serves as a pseudonym for the destination node, while the remaining ones (r_{ij} in the figures) enable backward routing on the destination chain.

Nodes store the ROUTINGIDS of the chains they belong to in a ROUTINGTABLE. With reference to Figure 2, let node p be a proxy in the chain of node b. p's ROUTINGTABLE contains an entry indexed by b's proxy ROUTINGID (p_b in the figure). This entry contains (i) p's secret key for the chain (s_p^b), (ii) the identifier of the

[4] We use uppercase characters for public keys and lowercase for private or secret keys.

previous node in the chain (v), (iii) the public chain key of v (C_v), and (iv) the ROUTINGID of the link between p and v (r_{pv}). Intermediate chain nodes also have an analogous entry in their routing tables, but indexed by the ROUTINGID of the link to the next node in the chain. Node v therefore has an entry indexed by r_{pv} and containing (i) v's secret key for the chain (s^b_v), (ii) the identifier of z, (iii) z's public chain key (C_z), and (iv) the ROUTINGID of the link to z (r_{zv}).

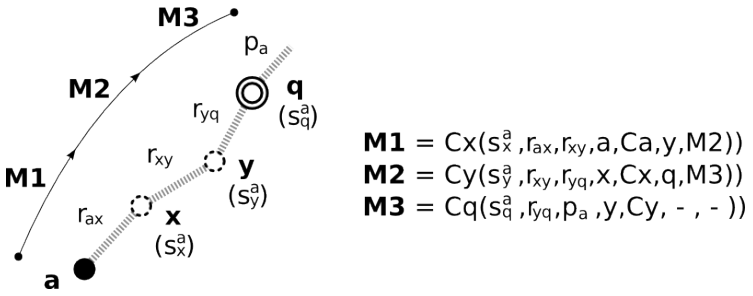


Fig. 1: *Proxy chain creation.*

3.2 FreeRec Three-Layer Architecture

Our goal in building proxy chains is to enable the architecture described in Section 2 to operate anonymously. To make this possible, we replace the two protocols of Section 2 with a three-layer architecture. We introduce a RPS protocol layer, which provides each node with a sample of the network from which to choose the members of its proxy chain. The RPS operates like a normal peer sampling protocol with one addition: it associates each node n with the information required for creating the chains. This comprises only the node's IP address, its public chain key C_n, and a timestamp. Interest profiles do not appear in the RPS views: they are protected by the anonymous PRPS layer.

The PRPS (Proxied Random Peer Sampling) uses the information provided by the RPS to build a proxy chain for each node. It then exploits these chains in gossip exchanges thereby providing each node with a random sample of anonymous nodes. In doing this, it also allows nodes to learn about the necessary information to route messages anonymously to other nodes. Consider a PRPS view containing an entry for node b. The entry does not include b's IP address and port. Rather, it is identified by b's proxy ROUTINGID (p_b). In addition, it contains the IP address and port of b's proxy (p), p's public chain key (C_p), b's public message key (K_b), and b's interest profile.

PRPS views allow nodes to learn about the anonymous information referring to another node without being able to associate it with the node's precise identity. Nodes exchange views like in a standard RPS. However, they channel all view exchanges through their proxy chains. The PRPS thus replaces the RPS protocol of the architecture of Section 2, thus enabling anonymous profile exchanges.

(C_n, c_n)	Chain key pair of node n	c_n private key, C_n public key
(K_n, k_n)	Message key pair of node n	k_n private key, K_n public key
s_n^x	Secret key generated by node x and shared with node n	
RPS	Random peers sampling	@ip, timestamp, C_n
PRPS	Random proxies sampling	@ip, timestamp, ROUTINGID, C_n, profile, K_n
CLUSTERING	Interest-based neighborhoods	@ip, timestamp, ROUTINGID, C_n, profile, K_n
RT	ROUTINGTABLE $[r_{pv}]$	s_v^b, z, C_z, r_{zv}

Table 1: *Data structures maintained on a node v, followed by p and preceded by z in b's chain.*

The PRPS serves as a basis for the top layer of our architecture: a CLUSTERING protocol, like the one in Section 2. However, unlike in Section 2, the clustering protocol also performs all its view exchanges using the proxy chains built by the PRPS layer. This allows our architecture to build decentralized personalized services in a completely anonymous manner.

3.3 Protocol Details

In the remainder of this section, we provide additional details about how the PRPS protocol manages chains and encrypted routing.

Building Proxy Chains. A node a can start building its proxy chain once its RPS view is filled with a random set of nodes. Specifically, a first determines how many other nodes should be in its chain by extracting a random number k from $\mathsf{ch_{min}}$ to $\mathsf{ch_{max}}$ included. Then it extracts k nodes from its RPS view and it sets the first extracted node as a proxy p and the remaining ones (if any) as intermediate nodes i in the order they were extracted. a builds a create-chain message as described on Figure 1.

The message consists of concentric *onion* layers. Each layer is a CREATECHAIN message encrypted with the chain key of one of the nodes that will constitute the chain. The innermost message, $M3$ in the figure, is encrypted with the proxy's chain key and contains (i) the proxy's secret key (s_q^a), (ii) the proxy ROUTINGID for the chain (p_a in the figure), and (iii) the ROUTINGID for the link between the proxy and the last intermediate node (r_{yq} in the figure), (iv) the previous node's IP address and port (y), and (v) its public chain key (C_y).

After creating $M3$, the initiator creates a message for the last intermediate node in the chain (y in the figure). This message contains (i) the previously encrypted message for the proxy ($M3$), (ii) the next node's IP address (the IP address of the proxy q in this case) and port (q), (iii) node y's secret key (s_y^a), (iv) the ROUTINGID of the link between y and the next node in the chain (r_{yq}), (v) the ROUTINGID of the link between y and the previous node in the chain (r_{xy}), (vi) the previous node's IP address and port (x), and (vii) its public chain key (C_x).

The initiator encrypts $M2$ with y's chain key and then it repeats the process by adding a layer for each of the nodes it selected for its chain, the last of these being the one closest to the initiator itself, x in the figure. The initiator then sends the outermost message to this node initiating the chain-creation process.

Each node receiving a CREATECHAIN message decrypts it and uses its content to update the information in its routing table. It then forwards the encrypted inner-layer message to the next node in the chain, which operates analogously. The proxy performs the same operations except that it does not forward the message further. If a chain node is already part of another chain with the same ROUTINGID, it replies with an error message to the initiator, which will recreate the chain using a different ROUTINGID.

Sending Messages through Chains FreeRec achieves mutual anonymity: when two nodes exchange messages, both the sender and the receiver are anonymous. Nodes use their proxy chains to send and receive encrypted messages as part of the PRPS and CLUSTERING protocols. Consider the example in Figure 2. Node a is sending a message m to a node with proxy p (not knowing b's id), public message key K_b, and proxy ROUTINGID p_b. Node a will have discovered this node, which happens to be b, through PRPS or CLUSTERING exchanges. As a result, the association between b's identity and p, K_b or p_b is unknown to a as well as to every other node in the system. The process unfolds as follows.

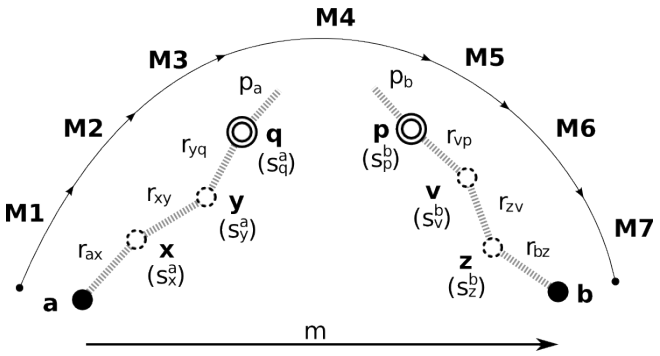


Fig. 2: *Message exchange between nodes a and b: a knows b's profile, the identity of p, but not the identity of b. Node b knows a's profile, the identity of q, but not the identity of a. Nodes in the chain cannot access a's or b's profile.*

First, a encrypts m using the destination node's public message key yielding $K_b(m)$. Then it prepares the first layer of its onion message. Specifically, a includes $K_b(m)$, and the destination's proxy incoming ROUTINGID, p_b in the figure. Then it encrypts the resulting message with the destination proxy's public chain key, yielding $M4$ in the figure. Node a continues the creation of the onion message by adding one layer from each of the nodes in its own chain, starting

from the proxy. The first of these layers, $M3$ is encrypted with the proxy's public chain key, and contains both $M4$ and the address of $M4$'s target: the destination node's proxy. The subsequent one, $M2$ contains $M3$ and the address of $M3$'s target, q in the figure. In general, consider a node n that is followed by a node m in a proxy chain. The corresponding onion layer will be encrypted with n's public chain key and will contain the IP address and port of m together with the immediate inner onion layer encrypted with m's public chain key. In the case of the figure, the outermost onion layer ($M1$) will be encrypted with node x's public chain key and will contain $M2$ and the IP address and port of node y.

After creating $M1$, node a sends it to x, which starts peeling off the first layer. It first decrypts the message using its private chain key and then forwards the contained encrypted message ($M2$) to the node indicated in $M1$ (y). Upon receiving the corresponding onion layer, each node in the chain proceeds analogously until the source node's proxy (q) forwards the innermost layer ($M4$) to the destination's proxy (p). This completes the first part of the routing process.

The destination's proxy (p) initiates the second part. It decrypts $M4$ and retrieves its content: a ROUTINGID, p_b, and an encrypted message for the destination node ($K_b(m)$). p first looks up p_b in its routing table and it retrieves (i) the associated secret key (s_p^b), (ii) the address and port of the previous node in the destination chain (v), and (iii) the ROUTINGID of the link leading to this node (r_{vp}). It then encrypts $K_b(m)$ using the retrieved secret key, yielding ($s_p^b(K_b(m))$). Finally it builds a message containing $s_p^b(K_b(m))$, and the ROUTINGID of the link to the previous node in the destination chain (r_{vp}). It encrypts this message using v's public chain key, yielding $M5$, and sends it to v.

When v receives $M5$, it decrypts it using its private chain key and retrieves $s_p^b(K_b(m))$ and the ROUTINGID of its link to p (r_{vp}). It looks up this ROUTINGID in its routing table and retrieves its own secret key s_v^b, the IP address and port of the previous node in the chain (z), z's public chain key, and the ROUTINGID of the link leading to it (r_{zv}). Node v encrypts $s_p^b(K_b(m))$ using s_v^b and places it in a message together with the retrieved ROUTINGID. It then further encrypts this message with z's public chain key and sends it to z. This process repeats at each of the intermediate nodes in the chain. Each adds an onion layer by encrypting the content of the message with its secret key and then wraps the result into a message with the routing information for the previous node in the chain.

When the destination node (b) receives the final message, it first decrypts it using its private chain key. Then it starts peeling off each of the onion layer added by the nodes in its proxy chain. To do so, it uses the secret keys it stored in its CHAINTABLE, starting with the one associated with the first intermediate node. After decrypting the layer added by its proxy, it obtains $K_b(m)$, which it further decrypts using its own private message key, ultimately retrieving the original message m.

Initialization For this process to work, the source of a message must not only have built its proxy chain, but it must also have the necessary information about the destination node. This consists of the destination node's public message key

(K_b), and of its proxy's IP address, public chain key (C_p), and ROUTINGID (p_b). During normal operation, nodes obtain this information through PRPS exchanges. However, this poses a problem during initialization when the PRPS view of a node is still empty.

Consider a node n with an empty PRPS view. The first time n establishes a proxy chain, it sends a PRPS view containing only its information to all the nodes in its RPS view. The corresponding messages go through the proxy chain of n until its proxy and then go directly to their targets. Consider a target node t receiving one such message. If t is a proxy for another node m, then it forwards the message to m along m's proxy chain. Otherwise t caches the message until it becomes a proxy for some other node. When a node m receives the initialization message forwarded by its proxy, it adds its content to its PRPS view.

In principle, the target node t could also add the information received from n to its own PRPS view. However, this would weaken the protocol's anonymity guarantees. An attacker n could send its entry to only one target node t. If it subsequently received a message from a proxy p, it could conclude that p is likely to be the proxy of t.

Changing proxy Nodes change their proxy chains periodically. This provides several benefits. It sustains anonymity over time by limiting the impact of attackers that may corrupt a node's proxy. It provides protection from attackers that may guess a node's keys. Moreover, it allows a node to react to path failures in its chain as a result of churn.

To change proxy chain, a node repeats the chain creation process every $t1$ time units. Once it has established a new anonymous path, it informs all the nodes in its PRPS and CLUSTERING view of its new proxy. To keep track of these changes, all proxies and intermediate nodes associate a timer $t2$ with each of the entries in their routing tables. When $t2$ expires, they delete the corresponding entry. Nodes choose the timer value so that $t2 > t1 + \delta$ where δ is an upper bound on the time required to create a chain.

After a node has set up a new chain, it initiates a PRPS exchange with all nodes in its PRPS view and a CLUSTERING exchange with all those in its CLUSTERING view. A node that receives a fresher PRPS entry with the same proxy ROUTINGID as an existing entry (*i.e.* entry pointing to the same destination node) updates this entry with the new proxy identifier, proxy chain key, message key, and profile.

An important side effect of changing proxies is that the minimum length of the chain $\mathsf{ch}_{\mathsf{min}}$ should be at least as large as 1. If $\mathsf{ch}_{\mathsf{min}} = 0$, then a node n would be its own proxy with probability $1/\mathsf{ch}_{\mathsf{max}}$. An attacker could easily exploit the fact that this is significantly larger than the probability of choosing a random proxy. For $\mathsf{ch}_{\mathsf{min}} \geq 1$, a node that serves in n's proxy chain for several times could still observe that n appears as a previous chain node more often than others. Yet, inferring this information would require n to choose the attacker as the first node in its chain for several times. This makes the attack for $\mathsf{ch}_{\mathsf{min}} \geq 1$ very unlikely to succeed in practice.

4 Evaluation

4.1 Experimental setup

We evaluated FreeRec by simulating its behavior and by deploying its implementation on PlanetLab in the context of a news-personalization use case. We combine FreeRec with a gossip-based dissemination protocol to recommend news items to a population of users. A user interest profile contains the news items she received and liked. When a user generates an item or expresses a positive opinion on a received item, she forwards it to her neighborhood in FreeRec's anonymous interest-based topology. Gossip frequency in all protocols is set to one per simulation cycle and of one every 2s in PlanetLab.

Dataset. We use a real dataset: we conducted a survey on around 250 news items (selected randomly from a set of RSS feeds on various topics). We exposed the item list to around 100 colleagues and relatives and gathered their reactions (like/dislike) to each news item. This provided us with a small but real dataset of users exposed to exactly the same news items. To scale our system, we generated 5 instances of each user and news item in the experiments. The resulting dataset gathers 1235 news items for 530 users. We inject each item into the system at a random time instant by selecting a random source node.

Metrics. We evaluate FreeRec along two metrics of performance and quality. Firstly we measure the overhead of the system in terms of the network traffic it generates. For simulations, we compute the total number of sent messages, the number of messages which have not reached its destination due to message loss and the number of hops for messages. For our PlanetLab deployment, we instead measure the average consumed bandwidth and the latency to receive a message. Secondly, to assess the impact of FreeRec on the quality of the recommendation, we compute *recall* and *precision*. Both measures are in $[0, 1]$. For an item, a recall of 1 means that all interested users have received the item. Yet, this measure does not account for spam since a trivial way to ensure a maximum recall is to send all news items to all users. This is precisely what precision accounts for. A precision of 1 means that the news item has reached only the users that are interested in it. An important challenge in information retrieval is to provide a good trade-off between these two metrics. This is expressed by the F1-Score, defined as the harmonic mean of precision and recall [25].

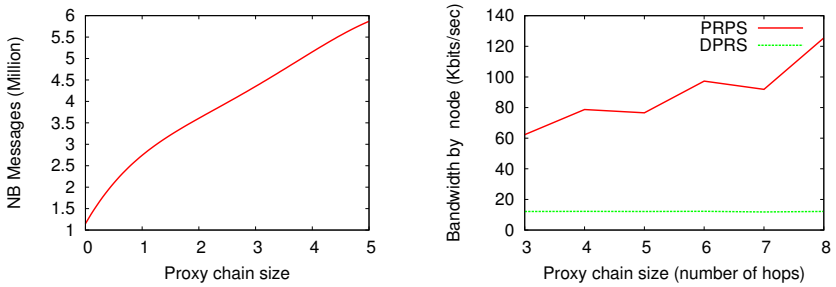
$$Precision = \frac{|\{interested\ \ users\} \cap \{reached\ \ users\}|}{|\{reached\ \ users\}|}$$

$$Recall = \frac{|\{interested\ \ users\} \cap \{reached\ \ users\}|}{|\{interested\ \ users\}|}$$

$$F1 - Score = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

4.2 Results

Overhead We start by considering the overhead of the proxy chain in terms of number of messages. Clearly the longer the chain, the more anonymous the system. This cost is a function of the length of the proxy chain: the more the intermediate nodes in the chain, the higher the cost. Figure 3 depicts the number of messages according to the size of the proxy chain with a neighborhood fixed to 25. Results (Fig. 3a) show that a chain with only one proxy without intermediate nodes (*i.e.* size=2) brings a three-fold increase in the number of messages with respect to a chain-less system (size=0). This is because a message needs to go through two proxies (*i.e.* 3 hops) to reach its destination. Further adding intermediate nodes in the proxy chain proportionally increases the number of hops and the number of messages. Fig. 3b shows the overhead in PlanetLab of the two protocols RPS and PRPS in terms of bandwidth consumption. We observe that the RPS overhead remains stable regardless of the size of the proxy chains for RPS exchanges carry only information about chain keys while PRPS carries the encrypted messages. For this reason, the cost of PRPS increases linearly with the length of the chain.[5]



(a) Simulation (b) Deployment

Fig. 3: *Overhead according to the size of the proxy chain, in function of number of messages and bandwidth consumption for simulation and PlanetLab deployment.*

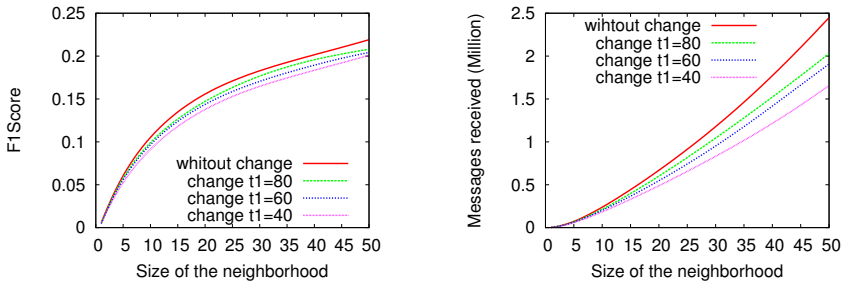
The impact of proxy changes To remain anonymous over time, nodes periodically renew their proxy chains. After setting up a new chain, a node advertises the information about its new proxy through PRPS and CLUSTERING exchanges. However, propagating this information takes time and some nodes only learn about the new proxy after several cycles. During this interval, a node that is unaware of the proxy change will send its messages to the old proxy. Consequently, messages will correctly go through the source node's possibly-new proxy chain, but they will reach the destination node's old proxy chain. If any of the nodes in this chain has already removed the corresponding entries from its routing table, it will silently discard the message.

As explained in Section 3.3, nodes remove entries from their routing tables δ time units after the creation of the new chain. During this time, the nodes in the

[5] CLUSTERING (not shown) has a similar behaviour as PRPS with a bandwidth consumption exactly twice as much as that of the PRPS due to the larger gossip size.

old chain can still forward information backwards towards the chain owner. This leaves some time for the propagation of the new chain's information, but it does not eliminate the possibility of losing messages. Figure 4 evaluates the impact of this aspect in the context of our news-dissemination testbed as a function of the size of the CLUSTERING view, with $\delta = 10$ cycles.

Figure 4a shows that the impact of message loss on the F1-Score is very limited. When nodes change proxy every 80 cycles (*i.e.* t1=80), performance is almost indistinguishable from the stable case where nodes keep the same proxy over the whole experiment. When the chain changes more frequently (smaller values of $t1$) the percentage loss in F1-Score is slightly higher, but it remains lower than 10%. Figure 4b completes these results by comparing the number of sent messages with those that are actually received. Clearly message loss increases with the frequency of proxy changes. When nodes change proxies every 40 cycles (*i.e.* three times in the experiment) the number of lost messages is one fourth of the total number of messages.



(a) F1-Score (b) Message loss

Fig. 4: *F1-Score and received messages for various* $t1$ *values (Simulations).*

Latency Figure 5 analyzes latency in our PlanetLab deployment (PL). The plot shows the time required by the PRPS protocol to establish a proxy chain, and by a message exchange that uses the chains both on the source and on the destination side. In the case of chain creation (CC), latency results from key generation, encryption/decryption operations, and message transmission. In the case of message exchanges (ME), there are only encryption/decryption operations and message transmission; yet messages have to travel for twice as many hops as in the case of chain creation.

The time required to create the proxy chain increases significantly with its size, while time required for exchanging messages increases only slightly. Moreover, creating the chain takes approximately two to three times as long as forwarding a message (40s vs 15s with 8-hop chains), even though forwarded messages have to travel for twice as many hops. This clearly shows that latency results mainly from computational cost. To understand the reasons for this seemingly poor performance, we ran the same test by instantiating all the nodes on a local server (LS). In this case, both operations complete in less than 3s, and exchanging messages does take longer than creating chains. This confirms that the

high latency exhibited in a PlanetLab setting results mainly from long processing times when performing cryptographic operations on overloaded machines.[6]

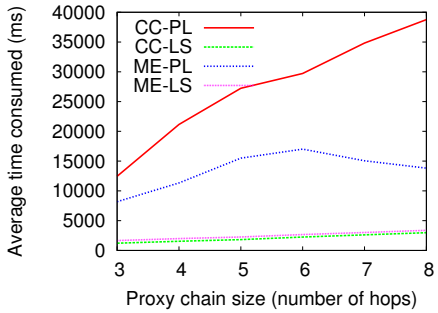


Fig. 5: *Latency of chain creation and message forwarding (deployment).*

5 Related Work

While personalization greatly enhances user experience, it raises privacy risks and concerns. Several collaborative-filtering approaches [20, 21] have tried to preserve the privacy of sensitive data address by applying randomized masking and distortion techniques to user profiles. However, [12, 16] show that privacy-sensitive information can be separated from such random distortion. To overcome this limitation, [7] uses noise that is not random but depends on the interest of users. This limits the amount of information exchanged between users to coarse-grained user profiles that only reveal the least sensitive information. Other decentralized approaches such as [9, 10, 19] exploit homomorphic encryption in a P2P environment. [13], in turn, addresses privacy by trust where participants exchange information and profile only across trusted content producer/consumer pairs. [2] proposes a differentially private protocol to measure the similarity between profiles. While differential privacy provides a strong notion of privacy, [18] highlights its important trade-off between privacy and accuracy.

A number of authors have proposed to address privacy by means of anonymity. Some, like [4] achieve receiver anonymity using group communication primitives like broadcasting, multicasting, and flooding. Others [11] focus on sender anonymity and relay messages from a node along a single anonymous path formed by nodes within the infrastructure.

Onion routing belongs to the latter group. It uses chains of router nodes that pack messages into onions: recursively encrypted data structures that contain the necessary routing information at each layer. When receiving an onion, a router removes a layer by decrypting it with its private key. At this point, it discovers either that it is the destination of the message, or the identity of the next router in the onion's forwarding path. . Tor [11] uses this model but cannot be readily integrated with decentralized personalization services.

[6] PlanetLab machines are notoriously overloaded, and the proximity of the SIGCOMM deadline might have resulted in even higher load.

Some authors have already suggested the integration of gossip and anonymous services. The work in [23] uses gossip protocols to improve the robustness of trust-based overlays to provide privacy-preserving data dissemination. More precisely, it creates and maintains additional anonymous links on top of an existing social overlay. Similarly, [22] relies on gossip protocols to supports confidential communications and private group membership. This solution leverages existing multi-hops paths to circumvent network limitations such as NAT and firewalls to form anonymous channel. Neither however combines anonymity with personalization. Gossple [5] does this to some extent and builds a fully decentralized anonymous collaborative network. Its gossip-on-behalf protocol hides the association between a user and her profile. Yet, in Gossple, a proxy controls some of the node's data structures. This is a significant disadvantage if the proxy wishes to censor specific information. In FreeRec, on the other hand, a proxy can at most drop messages randomly as it has no way to access their content.

Other works on gossip-based protocols have focused on tolerating byzantine faults such as BAR gossip [17], the secure peer sampling [15], Brahms [6] or PuppetCast [3]. In this work, we do not consider that nodes can act as active adversary by operating maliciously in the protocol. In case of malicious nodes cheating in the protocol, FreeRec could leverage one of these solutions to tolerate byzantine nodes. Finally, some authors [29, 28] have suggested to address churn by replacing each onion router with a group of nodes. Such a technique could easily be integrated with our solution.

6 Conclusions

We presented FreeRec, a decentralized architecture for building anonymous personalized services. FreeRec equips nodes with bidirectional onion-routing-like proxy chains that allow nodes to exchange their interest profiles without ever revealing their identities. FreeRec's core consists of three layers of gossip protocols. The bottom one is a standard random-peer-sampling protocol that provides nodes with the necessary information to build their proxy chains. The middle layer, the prps, constitutes the main contribution of this work and is an augmented RPS protocol: it builds and maintains proxy chains and uses them to provide each node with a continuously changing anonymous sample of the network. The top layer completes the picture by providing each node with a cluster of anonymous interest profiles that most closely resemble its own.

References

1. Anonymous surfing solution `http://anonymouse.org/`.
2. M. Alaggan, S. Gambs, and A-M. Kermarrec. BLIP: Non-interactive Differentially-Private Similarity Computation on Bloom Filters. In *SSS*, 2012.
3. Arno Bakker and Maarten van Steen. Puppetcast: A secure peer sampling protocol. In *EC2ND*, 2008.
4. N. Bansod, A. Malgi, B. K. Choi, and J. Mayo. Muon: Epidemic based mutual anonymity in unstructured p2p networks. *Comput. Netw.*, 2008.

5. M. Bertier, D. Frey, R. Guerraoui, A.M. Kermarrec, and V. Leroy. The gossple anonymous social network. In *Middleware*, 2010.
6. Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: byzantine resilient random membership sampling. In *PODC*, 2008.
7. A. Boutet, D. Frey, R. Guerraoui, A. Jegou, and A.-M. Kermarrec. Privacy-preserving distributed collaborative filtering. In *Activity Report*, 2013.
8. A. Boutet, D. Frey, R. Guerraoui, A. Jegou, and A.-M. Kermarrec. Whatsup decentralized instant news recommender. In *IPDPS*, 2013.
9. J. Canny. Collaborative filtering with privacy. In *SP*, 2002.
10. J. Canny. Collaborative filtering with privacy via factor analysis. In *SIGIR*, 2002.
11. R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *USENIX Security Symposium*, 2004.
12. Z. Huang, W. Du, and B. Chen. Deriving private information from randomized data. In *SIGMOD*, 2005.
13. S. Isaacman, S. Ioannidis, A. Chaintreau, and M. Martonosi. Distributed rating prediction in user generated content streams. In *RecSys*, 2011.
14. M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M.v. Steen. Gossip-based peer sampling. *TOCS*, 2007.
15. Gian Paolo Jesi, Alberto Montresor, and Maarten van Steen. Secure peer sampling. *Comput. Netw.*, 2010.
16. H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *ICDM*, 2003.
17. H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. Bar gossip. In *OSDI*, 2006.
18. A. Machanavajjhala, A. Korolova, and A. D. Sarma. Personalized social recommendations: accurate or private. *VLDB*, 2011.
19. B. N. Miller, J. A. Konstan, and J. Riedl. Pocketlens: toward a personal recommender system. *TOIS*, 2004.
20. H. Polat and W. Du. Privacy-preserving collaborative filtering using randomized perturbation techniques. In *ICDM*, 2003.
21. H. Polat and W. Du. Svd-based collaborative filtering with privacy. In *SAC*, 2005.
22. V. Schiavoni, E. Riviere, and P. Felber. Whisper: Middleware for confidential communication in large-scale networks. In *ICDCS*, 2011.
23. A. Singh, G. Urdaneta, M. van Steen, and R. Vitenberg. Robust overlays for privacy-preserving data dissemination over a social graph. In *ICDCS*, 2012.
24. X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009.
25. C. J. van Rijsbergen. *Information retrieval*. Butterworth, 1979.
26. S. Voulgaris, D. Gavidia, and M. v. Steen. Cyclon: inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 2005.
27. S. Voulgaris and M. v. Steen. Epidemic-style management of semantic overlays for content-based searching. In *Euro-Par*, 2005.
28. Y. Zhu and Y. Hu. Tap: A novel tunneling approach for anonymity in structured p2p systems. In *ICPP*, 2004.
29. L. Zhuang, F. Zhou, B. Y. Zhao, and A. Rowstron. Cashmere: resilient anonymous routing. In *NSDI*, 2005.